

Глава 13. Печать в Delphi.....	300
13.1 Объект TPrinter .....	301
13.2 Получение информации об установленном принтере.....	303
13.3 Текстовая печать. ....	306
13.4 Печать содержимого формы. ....	307
13.5 Вывод на печать изображения. ....	312



## Глава 13. Печать в Delphi

Число 13 действительно не счастливое. Хотя я и не суеверный и не верю в приметы, но именно 13-ю главу я переделывал несколько раз. Сначала она должна была начать рассказ про базы данных. Потом я понял, что поспешил и сдвинул базы данных на более поздний этап, а 13-ю главу переделал в графику. Но не тут-то было. До баз данных надо ещё рассказать про печать в Windows. Пришлось опять немного корректировать план, и 13-я глава получила новое название – «Печать в Delphi».

Но и на этом цифра 13 не закончила свои издевательства. Когда я написал свой первый пример для книги и захотел его протестировать, то вспомнил, что на моём HP400 в картридже закончились чернила. Заправлять картридж уже не имело смысла, потому что он уже несколько раз испытал на себе эту процедуру, да и покупать новый тоже не выгодно (слишком дорого). Поэтому я давно задумал купить новый принтер, но никак не доходили до этого руки. После этого пришлось срочно бежать в магазин и покупать новый принтер, потому что доверять своим знаниям хорошо, а всё же проверять желательно любые примеры. Я человек, а значит мне свойственны ошибки даже в простых примерах (ну бывает не доглядел что-то).

Почему я говорю «Печать в Delphi», а не «Печать в Windows». Во-первых – потому что Delphi сильно упрощает не только сам принцип печати, но и доступ к её возможностям.





### 13.1 Объект TPrinter

**П**ечать необходима везде и всегда. Ещё со времён первых компьютеров разработчики задумались об устройстве, которое могло бы выводить результаты расчётов на бумагу, чтобы не приходилось переписывать их ручками с монитора. Так и появился принтер, который очень сильно изменился за всё время своего существования, но не потерял актуальности.

Читать с монитора не всегда удобно, да и глаза устают от длительного чтения. Лично я всегда распечатываю всё необходимое для чтения на принтере, а потом изучаю в спокойной обстановке. Когда я путешествую в инете и вижу что-то интересное, то сразу отправляю на принтер. Так я экономлю не только время пребывания в сети, но и зрение.

Ну а офисные приложения вообще не возможно себе представить без возможности печати. Возьмём те же программы как Word и Excel. Да грош им цена, если они не смогут печатать созданные в них документы. Любые базы данных (о них мы поговорим в следующей главе) должны уметь выводить данные на принтер, формировать отчётность, которая так же должна быть доступна для печати и т.д.

Изначально в Windows задумывался очень удобный и простой механизм вывода информации – контекст устройства. Мы уже познакомились с графикой и увидели, как выводить графику на контекст графического устройства – монитор. Когда нужно вывести информацию на экран, нужно получить контекст рисования монитора (или видеокарты, кому как удобнее) и рисовать на нём. Для удобства в Delphi есть объект TCanvas, который упрощает доступ к функциям отображения данных.

Так вот, функции вывода на контекст устройства универсальны. Им всё равно, куда выводить данные – будь это контекст монитора или принтера или ещё какого-нибудь устройства отображения информации. Я думаю, что пока ещё не совсем ясно, о чём я говорю, но сейчас всё встанет на свои места.

В Delphi есть объект (обрати сразу внимание, что это не компонент), который содержит все необходимые для доступа к принтеру свойства и методы – **TPrinter**. Если посмотреть на эти свойства (чуть позже я опишу их), то сразу бросится в глаза свойство *Canvas* объектного типа **TCanvas**.

Да, это тот же самый объект **TCanvas**, который мы использовали при выводе графики. Я же говорил, что в объекте **TCanvas** собраны все необходимые функции для работы с графикой. Я так же сказал, что этим функциям абсолютно параллельно, куда выводить данные – на принтер или на экран монитора. Вот именно поэтому все компоненты способные отображать графику содержат в себе объектное свойство **TCanvas** и объект **TPrinter** способный выводить информацию на принтер тоже работает через него.

Раз у принтера есть такое свойство *Canvas*, то значит информацию, которую мы хотим вывести на печать надо выводить в виде графики. Даже тест выводиться именно как графика.

Немного теории закончено, теперь давай познакомимся с самим объектом **TPrinter** и посмотрим на его свойства и методы.

Свойства объекта **TPrinter**:

*Aborted* – переменная типа *Boolean*. Если она равна true, то пользователь прекратил вывод информации на принтер.

*Canvas* – объект типа **TCanvas**. Это холст, на который можно выводить информацию в графическом виде.

*Copies* – количество копий документа необходимых для печати.

*Fonts* – список шрифтов поддерживаемых принтером.

*Handle* – здесь храниться контекст принтера. Его ты можешь использовать, когда захочешь воспользоваться напрямую функциями WinAPI. Лично у меня такой надобности пока не было, потому что в объекте **TPrinter** уже есть всё необходимое.

*Orientation* – ориентация страницы. Это свойство может иметь одно из следующих значений: *poPortrait* – книжный или *poLandscape* – альбомный.

*PageHeight* – высота страницы в пикселях.

*PageWidth* – ширина страницы в пикселях.

*PageNumber* – номер печатаемой сейчас страницы.

*PrinterIndex* – число, которое указывает на номер выбранного сейчас принтера.

*Printers* – список типа **TStrings** установленных в системе принтеров.

*Printing* – если это свойство равно *true*, то принтер в данный момент печатает.

*Title* – заголовок или просто текстовое описание печатаемого документа. Этот заголовок будет отображаться во время печати в менеджере печати.

Со свойствами покончено, теперь давай разберёмся с методами объекта **TPrinter**:

*Abort* – прерывает текущую печать.

*BeginDoc* – начало печатаемого документа.

*EndDoc* – конец документа.

*GetPrinter* – получить индекс текущего принтера.

*NewPage* – новая страница документа.

*Refresh* – обновить информацию о шрифтах и принтерах.

*SetPrinter* – установить текущий принтер.

Чтобы объект **TPrinter** стал доступным твоему проекту ты должен добавить в раздел **uses** модуль *Printers*. Объект **TPrinter** не надо инициализировать. Достаточно только подключить модуль и объект становится доступным через переменную *Printer*. Это ты увидишь в примерах этой главы.

Но прежде чем переходить к практике я должен сделать ещё несколько замечаний. Я говорил, что печать похожа на отображение информации на экране и на низком уровне используются одни и те же API функции. Я от своих слов не отказываюсь, но всё же прежде чем печатать, тебе нужно учитывать следующие особенности контекста печати:

1. Если с экрана можно стереть информацию, то изображение, выведенное на Canvas объекта принтера затереть уже не возможно после начала процесса печати.

2. Принтеры имеют большее разрешение, чем экран. Самые простейшие принтеры сейчас могут печатать графику с разрешением до 1200 точек на дюйм. Если ты попытаешься вывести картинку размером 200x200 на современный принтер, без каких либо корректировок, то пользователь этого изображения просто не увидит.

3. Не все принтеры одинаково работают с графикой, поэтому нужно давать пользователю возможность выбирать качество печати. Желательно всегда перед выводом на печать отображать стандартное окно настроек печати. Если на экран вывод производится только один раз, то на принтере может понадобиться несколько копий. Это замечание не является обязательным, но советую учитывать это, чтобы твоя программа соответствовала признакам хорошего тона. А вдруг пользователю нужно 100 копий, так что ему теперь сто раз нажимать кнопку печати?

4. Ты должен давать возможность прервать печать в любой момент. Например, это может понадобиться, когда закончилась бумага, и её больше нет. Ты же не сможешь заблокировать работу программы, пока пользователь не сбегает в магазин. Конечно же такую возможность всегда предоставляет драйвер печати, но и ты не забывай о хорошем тоне.

### 13.2 Получение информации об установленном принтере.

Сейчас мы напишем программу, которая пока ещё не будет печатать, но зато она будет вытаскивать из системы информацию об установленных принтерах. Я сделаю её минимально простой, только чтобы ты понял основу работы с объектом *TPrinter*.

Создай новый проект и сразу же допиши в разделе **uses** модуль *Printers*, чтобы объект *TPrinter* стал доступным проекту.

Моя программа будет показывать текущий выбранный принтер и список всех доступных в системе устройств печати. Для этого на форме понадобится одна строка ввода *TEdit* для отображения текущего принтера и один список *TListBox* для отображения полного списка. У строки ввода можно установить свойство *ReadOnly* в *true*, потому что информация из этой строки только для отображения и не должна редактироваться, да и смысла нет её редактировать.

На рисунке 13.2.1 показана форма моей будущей программы. Ты можешь расположить компоненты по-другому, но я предпочитаю именно такое расположение. Главное делай подробные подписи для выводимой информации, иначе программа становится понятной только тебе, и больше уже никому не понадобится.

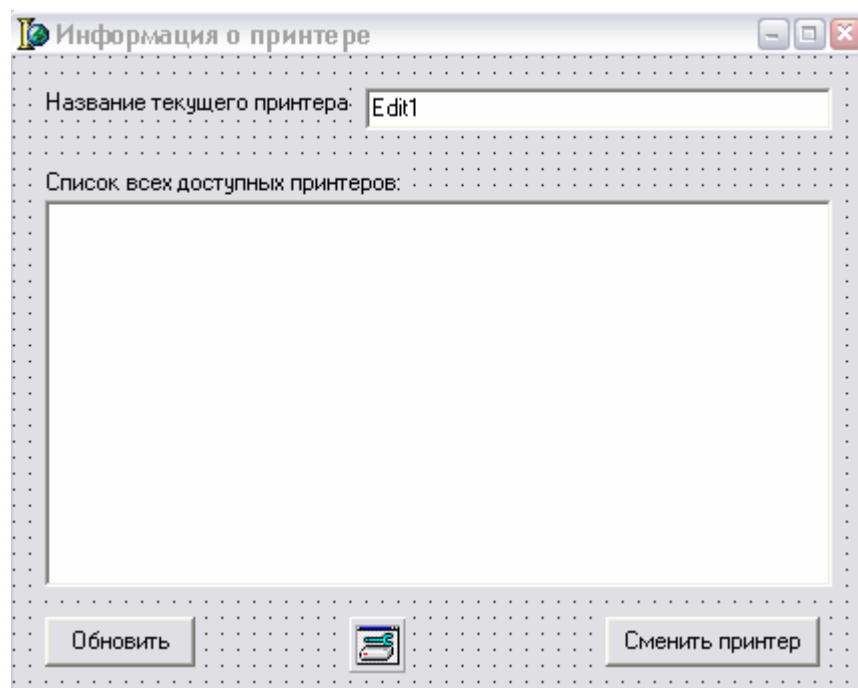


Рисунок 13.2.1 Форма будущей программы

Теперь брось на форму две кнопки: «Обновить» (для обновления информации о принтерах) и «Сменить принтер» (понятно зачем).

Нам так же понадобится компонент *PrinterSetupDialog* с закладки *Dialogs* палитры компонентов. Этот компонент предназначен для отображения стандартного окна настроек принтера.

По нажатию кнопки «Обновить» пишем следующий код:

---

```
procedure TForm1.Button1Click(Sender: TObject);  
var
```

```
i:Integer;  
begin  
  ListBox1.Items.Clear;  
  for i:=0 to Printer.Printers.Count-1 do  
    ListBox1.Items.Add(Printer.Printers.Strings[i]);  
  
  Edit1.Text:= Printer.Printers.Strings[Printer.PrinterIndex];  
end;
```

---

В первой строке кода я очищаю текущее содержимое списка *ListBox1*. Потом запускаю цикл от 0 до количества установленных в системе принтеров минус 1 (количество принтеров находится в *Printer.Printers.Count*). Свойство *Printers* объекта *Tprinters* имеет тип *TStrings*, как свойство *Items* компонента *TListBox*, поэтому их свойства и методы одинаковы и мы с ними уже не раз работали.

Почему я отнимаю единицу? Да потому что цикл начинаю с нуля. Допустим, что у тебя установлено 2 принтера и в переменной *Printer.Printers.Count* будет находиться 2. В этом случае цикл будет выполняться от 0 до 2, т.е. три раза для 0, 1 и 2. Но принтеров только 2, поэтому нужно отнять единицу.

Внутри цикла я добавляю в список названия принтеров. Названия хранятся в переменной *Printer.Printers.Strings[i]*, где *i* – это индекс принтера, изменяющийся от 0 до значения из *Printer.Printers.Count* минус 1.

После этого я вывожу в строку ввода название текущего принтера. Индекс текущего принтера – это *Printer.PrinterIndex* значит название текущего принтера можно получить так: *Printer.Printers.Strings[Printer.PrinterIndex]*.

Эту же процедуру можно вызвать по событию *OnCreate* или *OnShow* главной формы, чтобы после старта программа сразу же получала необходимую информацию:

---

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  Button1Click(nil);  
end;
```

---

Нам осталось только написать код для кнопки «Сменить принтер» и программа готова. Итак, по нажатию этой кнопки пишем коротко и ясно:

---

```
procedure TForm1.Button2Click(Sender: TObject);  
begin  
  PrinterSetupDialog1.Execute;  
  Button1Click(nil);  
end;
```

---

В первой строке я просто отображаю окно *PrinterSetupDialog1* (окно настроек принтера). Во второй строке я вызываю процедуру, которую мы написали для обновления информации о принтерах.

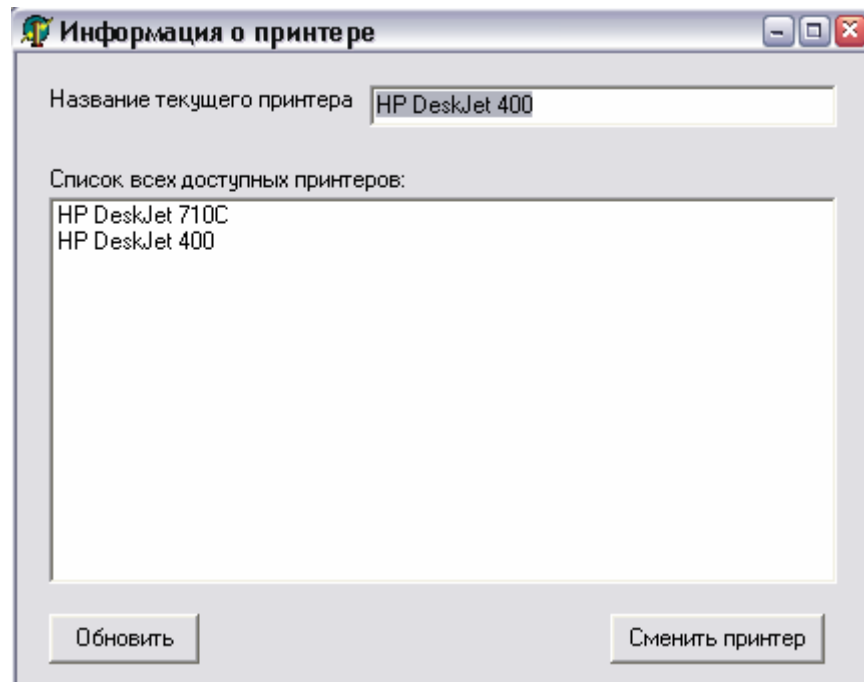


Рисунок 13.2.2 Результат работы программы

Теперь запусти программу и посмотри на результат. Если у тебя в системе только один принтер (ну нет у тебя больше принтеров) или вообще нет ни одного, то зайди в «Панель управления» Windows, выбери там «Принтеры» и установи пару любых принтеров вручную. Система от этого работать хуже не будет, зато ты сможешь полноценно протестировать этот пример.

Нажми на кнопку сменить принтер и перед тобой откроется окно похожее на рисунок 13.2.3. В этом окне, в выпадающем списке «Имя» измени имя текущего принтера на другое и закрой окно кнопкой «ОК». Обрати внимание, что в твоей программе, в строке «Название текущего принтера» название автоматически изменилось. Это потому что мы после отображения окна сразу перечитали свойства принтеров. А они изменились, потому что окно *PrinterSetupDialog1* тесно связано с системой и автоматически обновляет всю информацию в объекте *TPrinter*. Поэтому нам не надо самостоятельно читать данные, изменённые в окне свойств принтера, и переносить в объект печати.

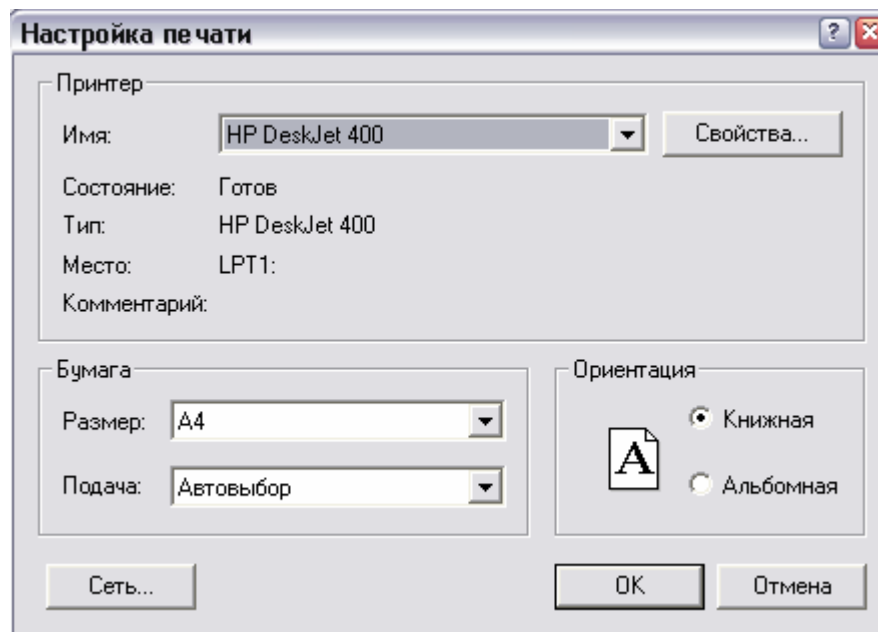



Рисунок 13.2.3 Окно свойств принтера

Советую так же обратить внимание на то, что *Printer* – это объект типа *TPrinter*, а я всегда говорил, что все объекты должны создаваться с помощью их же метода *Create* и уничтожаться с помощью метода *Free*. В данном случае этого делать не надо, всё делается автоматически. Именно поэтому я даже не показывал тебе метода *Create* и *Free*, когда расписывал объект *TPrinter*. Вот это тебе обязательно надо помнить!!!

 На компакт диске, в директории \Примеры\Глава 13\Свойства принтера ты можешь увидеть пример этой программы.

### 13.3 Текстовая печать.

**Х**отя нынешние принтеры и графические, но вывод на печать в виде текста ещё возможен. Представь себе, как сложно было бы жить, если бы нельзя было вывести содержимое компонента *TMemo*, как текст. В этом случае нам пришлось бы рисовать каждую строку в контексте рисования принтера (рисовать на *Canvas*-е с помощью его метода *TextOut*) при этом учитывая расстояния между строками.

Но есть ещё бог на свете и нам не надо так сильно мучиться. С принтером можно работать как с простым текстовым файлом.

Для открытия принтера в текстовом режиме используется процедура *AssignPrn*. В качестве единственного параметра этой процедуре надо передать переменную типа **TextFile**. После этого переменной назначен принтер по умолчанию. Дальше его нужно открыть с помощью процедуры *Rewrite*.

Как только файл открыт, в него можно печатать с помощью процедуры *writeln*, у которой два параметра:

1. Переменная типа **TextFile**, которой назначен принтер.
2. Текст, который надо распечатать.

После печати переменную надо освободить (закрыть файл, ассоциированный с принтером) с помощью процедуры *CloseFile*.

Вот простейший пример вывода текста «Hello world» в виде строки на принтер:



```
var
  f:TextFile;
begin
  AssignPrn(f);

  try
    Rewrite(f);
    Writeln(f, 'Hello world');
  finally
    CloseFile(f);
  end;
end;
```

---

В первой строке кода я назначаю переменной *f* принтер. После этого открытие файла ассоциированного с принтером и вывод на печать я заключаю между *try* и *finally*. Это необходимо, потому что если после выполнения процедуры *AssignPrn* переменной *f* не будет назначен принтер (ну нету его в системе, не установлен или вообще отсутствует), то при попытке открыть файл или начать печать произойдёт ошибка.

Между *finally* и *end* (код написанный здесь будет выполняться всегда, вне зависимости от того, была ошибка или нет) я закрываю открытый файл. Если бы я не ставил *try...finally..end*, а во время печати произошла ошибка, то файл, ассоциированный с принтером, остался бы открытым. А это значит, что последующая, нормальная работа принтера уже не гарантируется.

Вот ещё пример вывода на притер содержимого компонента *TMemo*.

---

```
var
  f:TextFile;
  i:Integer;
begin
  AssignPrn(f);

  try
    Rewrite(f);
    for i:=0 to Memo1.Lines.Count-1 do
      Writeln(f, Memo1.Lines.Strings[i]);
    finally
      CloseFile(f);
    end;
  end;
```

---

Попробуй сам разобраться, как работает этот пример. Для этого тебе нужно вспомнить свойства компонента *TMemo* и как работать с ним.

### 13.4 Печать содержимого формы.

**М**ы научились получать информацию о принтере и печатать в текст, теперь пора научиться выводить на печать графику. Хотя сейчас я покажу простейший пример печати, зато в нём будет всё необходимое тебе в будущем при построении сложных сцен.

Этот пример я взял из файла помощи Delphi и просто преподнес тебе в готовом виде. Это достаточно хороший пример, на котором можно научиться основам печати, а уже следующий пример, мы напишем более сложным с более качественной печатью.

Для примера нам понадобится форма, на которой будет расположен один компонент *TPageControl*. На нём можно создать несколько закладок (я создал две) и поместить на закладки различные компоненты. Я на первую закладку бросил текст и несколько компонентов *TShape*. На второй закладке я расположил картинку *TImage*. На печать я буду выводить содержимое закладок компонента *TPageControl* вместе с компонентами и картинкой. Каждая закладка будет печататься как отдельная страница.

Ну и напоследок брось на форму кнопку «Печать» и компонент *PrintDialog* с закладки *Dialogs* палитры компонентов. Вторым компонент предназначен для отображения стандартного окна запуска печати (рисунок 14.4.2). Окно печати похоже на окно настроек печати *PrinterSetupDialog*, но имеет свои отличия. Именно такое окно ты видишь каждый раз, когда запускаешь печать в других программах, таких как MS Word, Excel и др.

Вид моей формы ты можешь увидеть на рисунке 14.4.1. Можешь сделать такую же, а можешь попробовать что-нибудь своё.

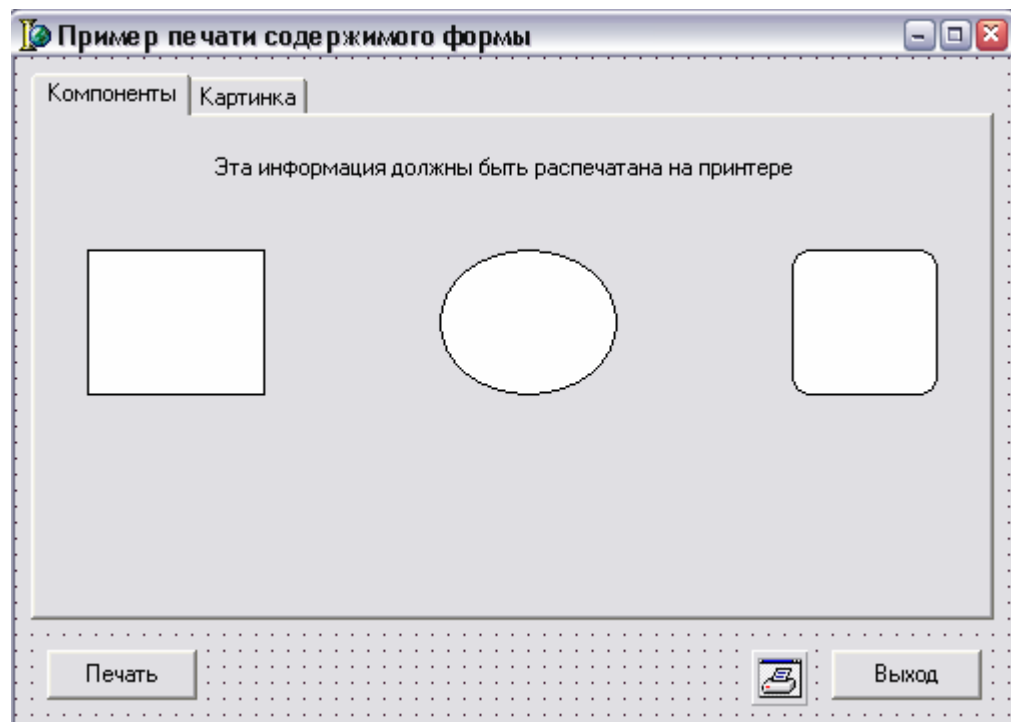


Рисунок 13.4.1 Форма будущей программы

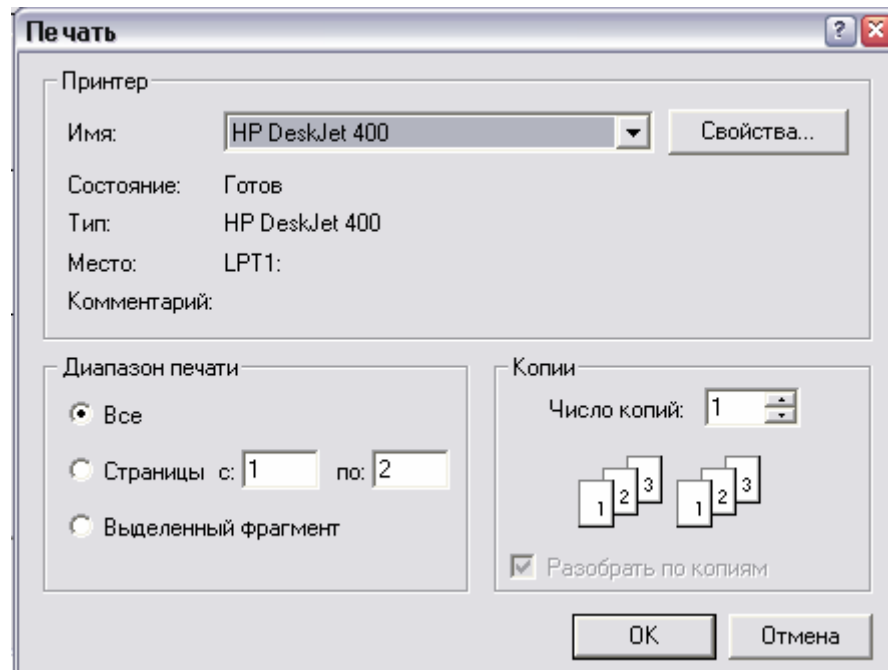


Рисунок 13.4.2 Окно запуска печати

Теперь перейдём к написанию кода. Создай обработчик события *OnClick* для кнопки печать и напиши в нём следующее:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i, Start, Stop: Integer;
begin
  PrintDialog1.Options := [poPageNums, poSelection];
  PrintDialog1.FromPage := 1;
  PrintDialog1.ToPage := PageControl1.PageCount;
  PrintDialog1.MinPage := 1;
  PrintDialog1.MaxPage := PageControl1.PageCount;
  if not PrintDialog1.Execute then exit;

  if PrintDialog1.PrintRange = prAllPages then
    begin
      Start := PrintDialog1.MinPage - 1;
      Stop := PrintDialog1.MaxPage - 1;
    end
  else //Если выбрано отличное от печати «Всё»
    if PrintDialog1.PrintRange = prSelection then
      begin
        Start := PageControl1.ActivePageIndex;
        Stop := Start;
      end
    else //Если выбрано отличное от «Выделенный фрагмент»
      begin
        Start := PrintDialog1.FromPage - 1;
        Stop := PrintDialog1.ToPage - 1;
      end;

  //Начало печати
  Printer.BeginDoc;
  for i := Start to Stop do
    begin
```

```
PageControl1.Pages[i].PaintTo(Printer.Handle, 10, 10);
if i <> Stop then
  Printer.NewPage;
end;
Printer.EndDoc;
end;
```

---

В первой строке кода я задаю опции окна печати *PrintDialog1.Options*. В этом свойстве храниться информация о том, что должно отображаться в окне «Печать». Возможны следующие значения:

*poDisablePrintToFile* – отключить возможность выбора печати в файл. Если ты добавишь в опции это значение, то пользователь не сможет выбрать «Печать в файл».

*poHelp* – показывать кнопку «Помощь» в окне печати.

*poPageNums* – разрешить пользователю выбирать диапазон печати (какие страницы нужно распечатать).

*poPrintToFile* – показывать в окне печати *CheckBox*, в котором можно выбрать печать в файл.

*poSelection* – давать возможность пользователю выбрать в окне печати печать выделенного фрагмента.

*poWarning* – показывать пользователю сообщения, если он пытается запустить работу на не установленный принтер.

Я в раздел настроек добавляю две опции *poPageNums* и *poSelection*:

```
PrintDialog1.Options := [poPageNums, poSelection]
```

Всё это можно было бы сделать проще: просто выделить компонент *PrintDialog1*, и в объектном инспекторе дважды щёлкнуть по свойству *Options*. После этого откроется список из всех этих свойств, где можно указать напротив необходимых свойств значение *true*. Я бы поступил именно так, но я же говорил, что пример взят из файла помощи по Delphi, а там настройки делали программно.

Во второй строке кода я присваиваю свойству *FromPage* компонента *PrintDialog1* начальное значение, с которого должна происходить печать. А в следующей строке я устанавливаю в свойстве *ToPage* количество печатаемых страниц. В это свойство я заново количество закладок у моего компонента *PageControl1*. Мы же договорились, что каждая закладка будет печататься на отдельной странице, значит, сколько закладок, столько и страниц будет печататься.

После этого я устанавливаю минимальное и максимальное значение страниц доступных для печати:

```
PrintDialog1.MinPage := 1;  
PrintDialog1.MaxPage := PageControl1.PageCount;
```

Все, настройки произведены, можно отображать окно печати:

```
if not PrintDialog1.Execute then exit;
```

Здесь используется конструкция *if not*, значит, если пользователь закроет окно через кнопку *Cancel*, то выполниться код написанный после *then*. А там у меня написан выход из процедуры – *exit*. Так что если не будет нажата в окне печати кнопка «OK», то процедура не будет выполняться дальше и печать не произойдёт. Ну а если нажат «OK», то код процедуры продолжит своё выполнение.

А дальше у меня идёт проверка, какой диапазон печати был выбран пользователем:

```
if PrintDialog1.PrintRange = prAllPages then
```

Этот код проверяет, если выбран диапазон печати всех страниц, то переменным *Start* и *Stop* будут присвоены значения минимального значения страниц и максимального значения соответственно, чтобы были распечатаны все закладки нашего компонента.

Если пользователь выбрал не все страницы, то нужно проверить, а может, он выбрал печать выделенного фрагмента?

```
if PrintDialog1.PrintRange = prSelection then
```

Если пользователь выбрал печать выделенного фрагмента, то переменным *Start* и *Stop* будет присвоено одно и то же - номер выделенной в данный момент закладки *PageControl1.ActivePageIndex*.

Ну и если пользователь не выбрал ни того, ни другого, значит, он выбрал начальную и конечную страницу, которые надо распечатать. В этом случае переменным *Start* и *Stop*, будут назначены значения выделенных пользователем страниц:

```
Start := PrintDialog1.FromPage - 1;  
Stop := PrintDialog1.ToPage - 1;
```

Обрати внимания, что я от выделенных пользователем страниц вычитаю единицу. Это потому что пользователь, как нормальный человек будет нумеровать страницы, начиная с единицы, а закладки нашего компонента *PageControl1* нумеруются с нуля.

Всё, теперь наши переменные *Start* и *Stop* содержат диапазон, который надо распечатать в зависимости от выбранных пользователем настроек и можно переходить к самой печати. Для начала распечатки нужно начать новый документ. Для этого нужно вызвать метод *BeginDoc* объекта *TPrinter*.

После этого запускаем цикл от стартовой страницы, до, последней выделенной:

```
for i := Start to Stop do
```


Внутри цикла выполняется следующий код:

```
PageControl1.Pages[i].PaintTo(Printer.Handle, 10, 10);  
if i <> Stop then  
Printer.NewPage;
```

В первой строке я заставляю прорисоваться очередную страницу на определённое устройство. Об этом говорит конструкция *PageControl1.Pages[i].PaintTo*. Метод *PaintTo* заставляет прорисоваться *i*-ю страницу на указанное устройство. Нужно устройство указывается в качестве первого параметра метода (здесь я указываю принтер). Остальные два параметра указывают отступ слева и сверху.

Далее я проверяю, если *i* не равна последней печатаемой страницы, то создать новую страницу, на которой будет распечатана следующая закладка. Если не производить этой проверки, то когда распечатается последняя страница, то будет создана новая, которая вылезет из принтера пустой. Это не ошибка, но будет не приятно, если из принтера в конце печати будет выползть пустой лист бумаги.

Самым последним методом вызывается *EndDoc* объекта *TPrinter* после чего принтер начинает печатать весь документ.

 На компакт диске, в директории \Примеры\Глава 13\Печать содержимого формы ты можешь увидеть пример этой программы.

### 13.5 Вывод на печать изображения.

В принципе, уже в прошлом примере я напечатал изображение, потому что я печатал закладки компонента *TPageControl* как самые настоящие картинки. Но пример был простой и не учитывал, что разрешение принтера выше, чем у монитора. Поэтому, если ты пытался распечатать примеры, то изображения получались слишком маленькими. В этой части я попытаюсь показать, как учитывать разрешение принтера.

Но сначала, давай взглянем на простейший пример вывода изображения на принтер:

---

```
begin
Printer.BeginDoc;
Printer.Canvas.Draw(10, 10, Image1.Picture.Bitmap);
Printer.EndDoc;
end;
```

---

Этот пример не отображает никаких диалогов, а просто начинает новый документ на принтере, потом копирует в его *Canvas* картинку из компонента *Image1* и заканчивает документ.

Это ещё один простейший пример, который не учитывает разрешение принтера. Он прост, но в реальных условиях не применим, потому что никому не нужно изображение, которое на бумаге меньше того, что видно на мониторе.

Теперь пора нам узнать, как определить разрешение принтера. Для этого используется WinAPI функция *GetDeviceCaps*, которая предназначена для получения информации о нужном устройстве. У этой функции два параметра:

1. Устройство, информацию которого мы хотим получить. Нам нужна информация о размере *Canvas* принтера, поэтому нужно будет передавать его указатель *Printer.Canvas.Handle*.

2. Какую именно информацию нам надо. Нам нужно количество пикселей по оси X и Y, поэтому будем указывать **LOGPIXELSX** (разрешение по оси X) и **LOGPIXELSY** (разрешение по оси Y).

Теперь реальный пример с использованием этой функции. Создай новый проект и брось на форму один компонент *TImage*, который будет хранить картинку для печати (сразу же загрузи туда любое изображение в формате bmp) и кнопка «Печать». На рисунке 13.5.1 ты можешь видеть форму моей будущей программы.

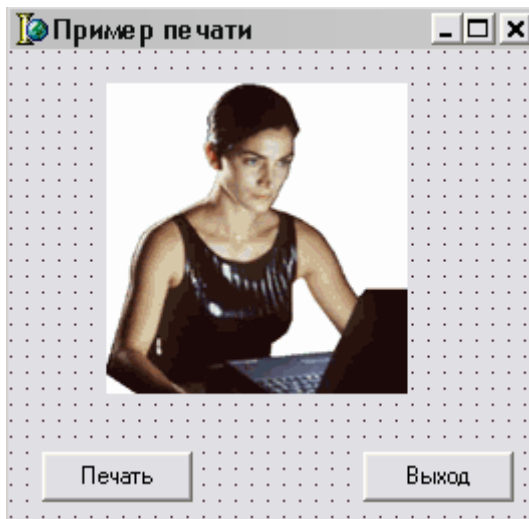


Рисунок 13.5.1. Форма будущей программы

Теперь создай обработчик события *OnClick* для кнопки и напиши там следующий код:

---

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X1,X2,Y1,Y2:Integer;
  PointsX,PointsY:double;
  PrintDlg:TPrintDialog;
begin
  // Создаю и отображаю на экране стандартное окно печати
  PrintDlg:=TPrintDialog.Create(Owner);
  if PrintDlg.Execute then
  begin
    //Начинаю новый документ
    Printer.BeginDoc;
    Printer.Canvas.Refresh;

    //Получаю информацию о разрешении принтера
    PointsX:=GetDeviceCaps(Printer.Canvas.Handle,LOGPIXELSX)/70;
    PointsY:=GetDeviceCaps(Printer.Canvas.Handle,LOGPIXELSY)/70;

    //Расчитываю размеры изображения
    X1:=round((Printer.PageWidth - Image1.Picture.Bitmap.Width*PointsX)/2);
    Y1:=round((Printer.PageHeight - Image1.Picture.Bitmap.Height*PointsY)/2);
    X2:=round(X1+Image1.Picture.Bitmap.Width*PointsX);
    Y2:=round(Y1+Image1.Picture.Bitmap.Height*PointsY);
    //Вывод изображения на печать
    Printer.Canvas.CopyRect(Rect(X1,Y1,X2,Y2),Image1.Picture.Bitmap.Canvas,
      Rect(0,0,Image1.Picture.Bitmap.Width,Image1.Picture.Bitmap.Height));
    Printer.EndDoc;
  end;
  //Уничтожаю созданное окно печати
  PrintDlg.Free;
end;
```

---

В самом начале я программно создаю компонент *PrintDialog* (стандартное окно печати), потому что на форму я его не бросал. Я сделал это специально, чтобы лишний раз показать тебе, как программно создаются и используются компоненты Delphi. Для

создания я выполняю код *TPrintDialog.Create(Owner)*, который выделит память под объект и возвратит мне ссылку на него. Эту ссылку я сохраняю в переменной *PrintDlg*, которая объявлена у меня в разделе **var** в виде переменной типа *TPrintDialog*.

После этого я показываю окно печати (*if PrintDlg.Execute then*) и если пользователь нажал «OK», то выполниться код между последующими *begin...end*.

Тут с первой строкой кода уже понятно – начало нового документа. После этого я обновляю всю информацию на холсте принтера (*Printer.Canvas.Refresh*).


Дальше я получаю информацию о разрешении принтера по вертикали и горизонтали с помощью функции *GetDeviceCaps*. Результат делится на 70 (просто коэффициент масштабирования, можешь подобрать любой подходящий тебе) и сохраняю в переменных *PointsX* и *PointsY*.

После этого идёт расчёт координат картинки. Я её постарался вывести по центру листа бумаги. Отступ слева я вычисляю по такой формуле: (ширина листа принтера – ширина картинки \* *PointsX*)/2. Отступ сверху вычисляю по следующей формуле (высота листа принтера – высота картинки \* *PointsY*)/2.

Отступы слева и сверху рассчитаны. Теперь нужно найти правую и нижнюю сторону изображения. Для этого я просто прибавляю к левой стороне ширину изображения (*Image1.Picture.Bitmap.Width\*PointsX*) и к верхней стороне высоту изображения (*Image1.Picture.Bitmap.Height\*PointsY*).

Теперь можно выводить картинку на холст принтера. Для этого я копирую изображение из *Image1* на холст принтера с помощью процедуры *Printer.Canvas.CopyRect*. Мы уже пользовались этой процедурой при работе с графикой и ты должен знать, что она умеет масштабировать копируемую картинку.

Всё, можно заканчивать документ (*Printer.EndDoc*) и уничтожать созданное окно (*PrintDlg.Free*). Хотя окно создано как локальное, потому объявлено внутри процедуры, и должно уничтожаться автоматически, я всё же делаю это, не надеясь на компилятор. Не устану напоминать, что все переменные объявленные как локальные (в разделе **var** внутри процедуры) инициализируются в стеке и автоматически уничтожаются сразу после выхода из процедуры. Но всё же те переменные, которым ты выделял память или создавал, как объекты нужно уничтожать самостоятельно, не надеясь на чистку стека. Ведь в стеке храниться только ссылка на объект или выделенную память, а сама память может быть выделена где угодно, но только не в этом стеке.

 На компакт диске, в директории \Примеры\Глава 13\Печать картинки ты можешь увидеть пример этой программы.